**Universidad de Oriente**

**Nucleó Anzoátegui**

**Escuela de Ingeniería y Ciencias Aplicadas**

**Departamento de Computación y Sistemas**

**Desarrollo de Software**



**UML 2.0 Completo**

**Estudiantes:**

Correa, Luis. C.I: 19.840.230 Ingeniería En Computación

**Profesor:**

Ing. Víctor Mujica.

**Barcelona, Julio de 2015**

**1.** **Metodología Orientada a Objetos (OOM)** 8

**2.** **Fases de OMT:** 9

2.1. Análisis. 9

2.2. Diseño del sistema. 9

2.3. Diseño de objetos. 9

2.4. Implementación. 10

**3.** **Modelos de OMT** 10

3.1. Modelo de objetos. 10

3.2. Modelo dinámico. 10

3.3. Modelo funcional. 10

**4.** **Ventajas de la Metodología Orientada A Objetos** 10

4.1. Reutilización. 10

4.2. Estabilidad. 10

4.3. Calidad. 11

4.4. Integridad. 11

**5.** **Lenguaje Unificado de Modelado (UML)** 11

**6.** **Para UML se considera que el modelo de proceso de software sea** 11

6.1. Basados en casos de uso 11

6.2. Centrado en la arquitectura 11

6.3. Iterativo 11

6.4. Incremental 11

**7.** **Ciclo de Vida de UML** 12

7.1. Iniciación o concepción. 12

7.2. Elaboración. 12

7.3. Construcción, 12

7.4. Transición. 12

**8.** **Modelo de la Arquitectura** 12

8.1. Se especifica la organización del sistema 13

8.2. Se seleccionan los elementos estructurales y sus interfaces a través de los cuales se constituye el sistema. 13

8.3. Se especifica el comportamiento, colaboraciones entre los componentes del sistema. 13

8.4. Se especifica la composición de los elementos estructurales y de comportamiento en subsistemas progresivamente más grandes. 13

8.5. Se especifica el estilo arquitectónico que guía la organización son: elementos estáticos, elementos dinámicos y sus interfaces, sus colaboraciones y su composición. 13

**9.** **Las vistas para describir mejor el sistema son** 13

9.1. La vista de casos de uso 13

9.1.1. Aspectos estáticos Diagramas de Casos de Uso 14

9.1.2. Aspectos dinámicos Diagramas de Interacción, Estado, Actividades 14

9.2. La vista de diseño 14

9.2.1. Aspectos estáticos Diagrama de Clases, Objetos 14

9.2.2. Aspectos dinámicos Diagramas de Interacción, Estado, Actividades 14

9.3. La vista de interacción 14

9.3.1. Igual que en la vista de diseño 14

9.4. La vista de implementación 14

9.4.1. Aspectos estáticos Diagrama de Componentes 14

9.4.2. Aspectos dinámicos Diagrama de Interacción, Estado, Actividades 14

9.5. La vista de despliegue 14

9.5.1. Aspectos estáticos Diagrama de Despliegue 15

9.5.2. Aspectos dinámicos Diagramas de Interacción, Estado, Actividades. 15

**10.** **Elementos del UML** 15

**11.** **Elementos Estructurales** 16

11.1. Clase 16

11.2. Interfaz 16

11.3. Colaboración 16

11.4. Casos de uso 16

11.5. Clases activas 16

11.6. Nodos 16

11.7. Artefactos 16

11.8. Componentes 16

**12.** **Elementos de Comportamiento** 16

12.1. Mensajes 16

12.2. Actividades 16

12.3. Estados 16

**13.** **Elementos de Agrupación** 16

13.1. Paquete 16

**14.** **Elementos de Anotación** 17

14.1. Nota. 17

**15.** **Elementos de Notación** 17

15.1. Nota. 18

15.2. Dependencia. 18

15.3. Clase. 18

15.4. Objetos. 18

15.5. Atributos. 18

15.6. Métodos. 18

15.7. Asociaciones. 18

15.8. Dependencias. 18

15.9. Responsabilidades. 18

15.10. Navegabilidad. 18

15.11. Composición. 18

15.12. Relaciones. 18

15.13. Herencia. 18

15.14. Generalización. 18

15.15. Actores. 18

15.16. Relaciones. 18

15.17. Caso de uso. 18

15.18. Línea de vida. 18

15.19. Activación. 18

15.20. Mensaje. 18

**16.** **Diagrama de caso de uso** 18

16.1. Actor 19

16.2. Relaciones 19

16.3. Caso de Uso 19

16.4. Límite de Sistema 20

16.5. Escenario 20

16.6. Extends 20

16.7. Include 21

**17.** **Diagrama de secuencia** 21

17.1. Línea de vida de un objeto 22

17.2. Objeto activo 22

17.3. Mensaje 22

17.4. Actor exterior 22

**18.** **Diagrama de Estados** 22

18.1. Estado 23

18.2. Transición 23

18.3. Evento disparador 23

18.4. Acción 23

18.5. Condición de guarda 23

**19.** **Diagrama de Actividades** 24

19.1. Actividad 24

19.2. División Sincronización 24

19.3. Transición de terminación 24

19.4. Unión Sincronización 24

19.5. Bifurcación o Reunificación 24

19.6. Calles y Flujo de objetos 24

**20.** **Diagrama de Componentes** 25

20.1. Interfaz 26

20.2. Puerto 26

20.3. Artefacto 26

20.4. Componentes 26

20.5. Paquetes o subsistemas 26

20.6. Relaciones de dependencia, generalización, asociación y realización 26

20.7. Nodo 26

**21.** **Diagrama de Ejecución** 26

1.1. Nodos 27

1.2. Componentes 27

**22.** **Diagrama de Clases** 28

1.3. Clase 28

1.4. Asociaciones 29

a) Asociación binaria 29

b) Sentido 30

c) Multiplicidad 30

d) Roles 30

1.5. Generalización 31

1.6. Clase paramétrica 31

1.7. Agregación 31

1.8. Composición 32

**23.** **Diagrama de Objetos** 33

23.1. Objetos 33

23.2. Enlaces 33

23.3. Clases 33

**24.** **Características de Diagrama de Objetos** 34

24.1. Los objetos se representan etiquetando el nombre de la instancia seguido de dos puntos (:) frente al nombre de la clase. 34

24.2. Se centra en comunicar un aspecto de la vista de diseño estática o la vista de procesos estática de un sistema. 34

24.3. Representa una escena de la historia representada por un diagrama de interacción. 34

24.4. Contiene solo aquellos elementos esenciales para comprender ese aspecto. 34

24.5. No es minimalista, no deja de informar al lector sobre la semántica importante. 34

24.6. Los diagramas de objetos `pueden contener notas y restricciones. 34

24.7. A veces se colocarán clases en los diagramas de objetos, cuando se quiera mostrar la clase que hay detrás de cada instancia. 34

**25.** **Técnicas comunes de modelado de Objetos** 34

25.1. Modelado de estructuras de objetos 34

25.2. Ingeniería Inversa 34

25.3. Modelado de Instancias discretas 35

**26.** **Clase Estructurada** 35

26.1. Modelado de una API 35

26.2. Para modelar una API 35

**27.** **Diagrama de Comunicación** 35

27.1. Objeto activo 36

27.2. Enlace 36

27.3. Mensaje 36

27.4. Multiobjeto 36

27.5. Objeto pasivo 36

27.6. Enlace transitorio 36

**28.** **Diagrama de Despliegue** 36

28.1. Componentes 37

28.2. Interfaz 37

28.3. Puerto 37

28.4. Artefacto 37

28.5. Paquetes o subsistemas 37

28.6. Relaciones de dependencia, generalización, asociación y realización 37

28.7. Nodo 37

**29.** **Diagrama de Paquetes** 38

29.1. Paquetes 38

29.2. Diagramas de UML internos 38

**30.** **Diagrama de Tiempo** 39

30.1. Mensaje 39

30.2. Objetos 39

**31.** **Diagrama de Visión Global de Interacciones** 40

31.1. Actividad 40

31.2. División Sincronización 40

31.3. Transición de terminación 40

31.4. Unión Sincronización 40

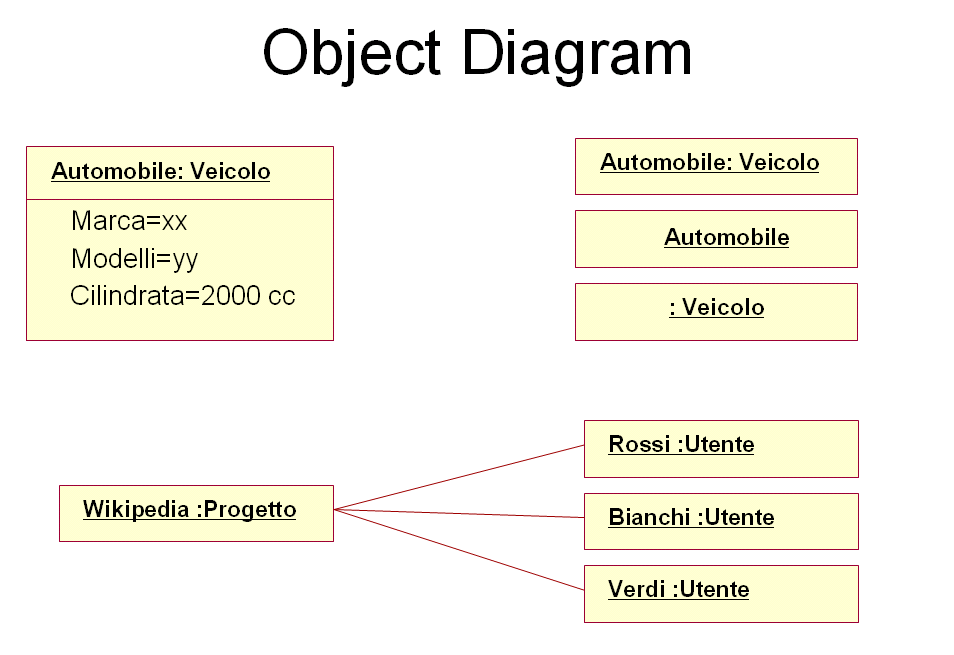
31.5. Bifurcación o Reunificación 40

31.6. Diagrama de Secuencia 40

## **Metodología Orientada a Objetos (OOM)**

En [UML](https://es.wikipedia.org/wiki/UML), diagrama que muestra una vista completa o parcial de los [objetos](https://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) de un sistema en un instante de ejecución específico.

Actualmente el modelo de objetos ha sido influenciado por un número de factores no sólo de la Programación Orientada a Objetos, POO. Además, el modelo de objetos ha probado ser un concepto uniforme en las ciencias de la computación, aplicable no sólo a los lenguajes de programación sino también al diseño de interfaces de usuario, bases de datos y arquitectura de computadoras por completo. La razón de ello es, simplemente, que una orientación a objetos nos ayuda a hacer frente a la inherente complejidad de muchos tipos de sistemas.

La metodología OMT (Técnica de Modelado de Objetos) fue creada por James Rumbaugh y Michael Blaha en 1991, es una de las metodologías de análisis y diseño orientados a objetos, más maduros y eficientes que existen en la actualidad. 

## **Fases de OMT:**

### Análisis.

El Analista Construye un modelo del dominio del problema, mostrando sus propiedades más importantes. El modelo de análisis es una abstracción resumida y precisa de lo que debe de hacer el sistema deseado y no de la forma en que se hará. Los elementos del modelo deben ser conceptos del dominio de aplicación y no conceptos informáticos tales como estructuras de datos. Un buen modelo debe poder ser entendido y criticado por expertos en el dominio del problema que no tengan conocimientos informáticos.

### Diseño del sistema.

**El Diseñador del Sistema** toma decisiones de alto nivel sobre la Arquitectura del Software. Durante esta fase el sistema se organiza en subsistemas basándose tanto en la estructura del análisis como en la arquitectura propuesta. Se selecciona una estrategia para afrontar el problema.

### Diseño de objetos.

**El Diseñador de Objetos** construye un modelo de diseño basándose en el modelo de análisis, pero incorporando detalles de implementación. El diseño de objetos se centra en las estructuras de datos y algoritmos que son necesarios para implementar cada clase. OMT describe la forma en que el diseño puede ser implementado en distintos lenguajes (orientados y no orientados a objetos, bases de datos, etc.).

### Implementación.

Los Objetos de Clases y relaciones desarrollados durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los principios de la ingeniería del software de forma que la correspondencia con el diseño sea directa y el sistema implementado sea flexible y extensible. No tiene sentido que utilicemos AOO y DOO de forma que potenciemos la reutilización de código y la correspondencia entre el dominio del problema y el sistema informático, si luego perdemos todas estas ventajas con una implementación de mala calidad.

## **Modelos de OMT**

* 1. Modelo de objetos.

Describe la estructura estática de los objetos del sistema. El modelo de objetos proporciona el entorno esencial en el cual se pueden situar el modelo dinámico y el modelo funcional. El objetivo es capturar aquellos conceptos del mundo real que sean importantes para la aplicación. Se representa mediante diagramas de objetos.

* 1. Modelo dinámico.

Describe los aspectos de un sistema que tratan de la temporización y secuencia de operaciones (sucesos que marcan los cambios, secuencias de sucesos, estados que definen el contexto para los sucesos) y la organización de sucesos y estados. Captura el control, aquel aspecto de un sistema que describe las secuencias de operaciones que se producen sin tener en cuenta lo que hagan las operaciones, aquello a lo que afecten o la forma en que están implementadas. Se representa gráficamente mediante diagramas de estado.

* 1. Modelo funcional.

Describe las transformaciones de valores de datos que ocurren dentro del sistema. Captura lo que hace el sistema, independientemente de cuándo se haga o de la forma en que se haga. Se representa mediante diagramas de flujo de datos

## **Ventajas de la Metodología Orientada A Objetos**

* 1. Reutilización.

Las clases están diseñadas para que se reutilicen en muchos sistemas.

* 1. Estabilidad.

Las clases diseñadas para una reutilización repetida se vuelven estables, de la misma manera que los microprocesadores y otros chips se hacen estables.

Se construyen clases cada vez más complejas. Se construyen clases a partir de otras clases, las cuales a su vez se integran mediante clases. Esto permite construir componentes de software complejos, que a su vez se convierten en bloques de construcción de software más complejo.

* 1. Calidad.

Los diseños suelen tener mayor calidad, puesto que se integran a partir de componentes probados, que han sido verificados y pulidos varias veces.

* 1. Integridad.

Las estructuras de datos (los objetos) sólo se pueden utilizar con métodos específicos. Esto tiene particular importancia en los sistemas cliente-servidor y los sistemas distribuidos, en los que usuarios desconocidos podrían intentar el acceso al sistema.

## **Lenguaje Unificado de Modelado (UML)**

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

## **Para UML se considera que el modelo de proceso de software sea**

* 1. Basados en casos de uso

Que los casos de uso sean un artefacto para establecer el comportamiento deseado del sistema, para validar la arquitectura, para las pruebas y para la comunicación entre las personas involucradas en el proyecto.

* 1. Centrado en la arquitectura

Que sea el artefacto básico para conceptuar, construir, gestionar y hacer evolucionar el sistema.

* 1. Iterativo

Que involucre la gestión del flujo de ejecutables del sistema.

* 1. Incremental

Donde cada nueva versión corrige defectos de la anterior e incorpora nueva funcionalidad. Un proceso iterativo e incremental se denomina dirigido por el riesgo, lo que significa que cada nueva versión se ataca y reducen los riesgos más significativos para el éxito del proyecto.

## **Ciclo de Vida de UML**

Durante el ciclo de vida de un proyecto software existen cuatro fases por las cuales debe pasar el proceso de desarrollo iterativo, entre las cuales se encuentran las siguientes:

* 1. Iniciación o concepción.

En ésta fase se desarrollan las ideas para otorgar un sistema final. Se determina lo que realizará el sistema, como estará compuesto arquitectónicamente, cual es el plan a seguir para lograrlo y además el costo que tendrá el mismo sistema, tomando en cuenta los riesgos que se puedan presentar durante la elaboración de la aplicación.

* 1. Elaboración.

Se deben determinar los requisitos del sistema y las pruebas sobre el mismo. En esta fase se establecen en detalle los casos de uso que se van a implementar para el desarrollo del programa y a la vez se diseña la estructura del sistema, teniendo al final de esta fase las actividades planeadas y los recursos necesarios para que se ejecute el proyecto.

* 1. Construcción,

Como su nombre lo dice, se comienza a construir la aplicación tomando en cuenta la estructura que se realizó del proyecto durante la etapa anterior. En esta fase se debe de tener un avance del sistema a crear, para que los usuarios otorguen opiniones de la aplicación y si existe algún error o deficiencia se puedan realizar los cambios necesarios para que este se desarrolle perfectamente.

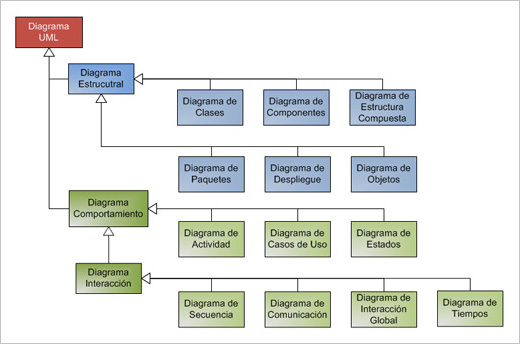
* 1. Transición.

El software es continuamente reexaminado y mejorado erradicando errores y añadiendo nuevas funcionalidades no contempladas. En esta parte de desarrollo se otorga el sistema a los usuarios para que se realicen pruebas del sistema, se pudiese llamar como si fuera una versión beta del programa, para si existiese fallos se puedan corregir con anterioridad y cumplir con el objetivo señalado en tiempo y forma.

La iteración afecta a estas cuatro fases. Una iteración es la repetición de una serie de instrucciones repetitivas que comprenden el proceso para lograr un sistema, como ejemplo de ello en programación se puede mencionar la recursividad, ya que esta permite repetir las instrucciones hasta que se cumpla una acción en específico. Este elemento está presente en UML como un conjunto bien definido de actividades, con un plan y unos criterios de evaluación, que acaban en una versión del producto.

## **Modelo de la Arquitectura**

La arquitectura es la que se encarga de describir la organización del sistema, selección de elementos estructurales y sus interfaces, el comportamiento, las colaboraciones entre los componentes del sistema, composición de los elementos estructurales



La arquitectura es un conjunto de decisiones significativas sobre

* 1. Se especifica la organización del sistema
  2. Se seleccionan los elementos estructurales y sus interfaces a través de los cuales se constituye el sistema.
  3. Se especifica el comportamiento, colaboraciones entre los componentes del sistema.
  4. Se especifica la composición de los elementos estructurales y de comportamiento en subsistemas progresivamente más grandes.
  5. Se especifica el estilo arquitectónico que guía la organización son: elementos estáticos, elementos dinámicos y sus interfaces, sus colaboraciones y su composición.

## **Las vistas para describir mejor el sistema son**

* 1. La vista de casos de uso

Comprende la descripción del comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de las pruebas y se utilizan los diagramas de casos de uso para capturar los aspectos estáticos mientras que los dinámicos son representados por diagramas de interacción, estados y actividades.

#### Aspectos estáticos Diagramas de Casos de Uso

#### Aspectos dinámicos Diagramas de Interacción, Estado, Actividades

* 1. La vista de diseño

Comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y de la solución. Esta vista soporta principalmente los requisitos funcionales del sistema, o sea, los servicios que el sistema debe proporcionar. Los aspectos estáticos se representan mediante diagramas de clases y objetos y los aspectos dinámicos con diagramas de interacción, estados y actividades.

#### Aspectos estáticos Diagrama de Clases, Objetos

#### Aspectos dinámicos Diagramas de Interacción, Estado, Actividades

* 1. La vista de interacción

Comprende los hilos y procesos que forman mecanismos de sincronización y concurrencia del sistema cubriendo el funcionamiento, capacidad de crecimiento y el rendimiento del sistema. Con UML, los aspectos estáticos y dinámicos se representan igual que en la vista de diseño, pero con el énfasis que aportan las clases activas, las cuales representan los procesos y los hilos.

#### Igual que en la vista de diseño

* 1. La vista de implementación

Comprende los componentes y los archivos que un sistema utiliza para ensamblar y hacer disponible el sistema físico. Se ocupa principalmente de la gestión de configuraciones de las distintas versiones del sistema. Los aspectos estáticos se capturan con los diagramas de componentes y los aspectos dinámicos con los diagramas de interacción, estados y actividades.

#### Aspectos estáticos Diagrama de Componentes

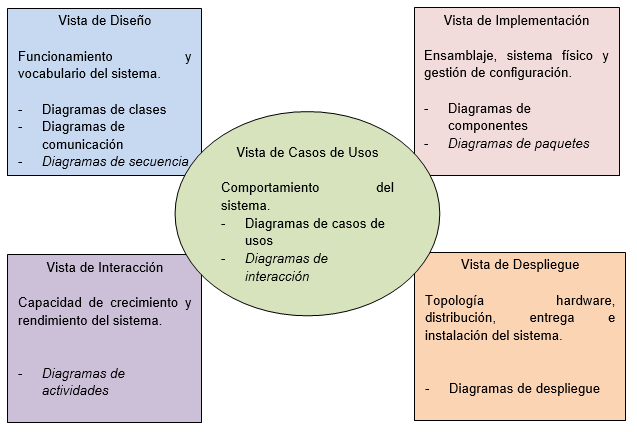
#### Aspectos dinámicos Diagrama de Interacción, Estado, Actividades

* 1. La vista de despliegue

De un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Se preocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema. Los aspectos estáticos de esta vista se representan mediante los diagramas de despliegue y los aspectos dinámicos con diagramas de interacción, estados y actividades.

#### Aspectos estáticos Diagrama de Despliegue

#### Aspectos dinámicos Diagramas de Interacción, Estado, Actividades.



## **Elementos del UML**

Hay 4 tipos de elementos en UML, elementos estructurales, elementos de comportamiento, elementos de agrupación y elementos de anotación.

## **Elementos Estructurales**

* 1. Clase
  2. Interfaz
  3. Colaboración
  4. Casos de uso
  5. Clases activas
  6. Nodos
  7. Artefactos
  8. Componentes

## **Elementos de Comportamiento**

* 1. Mensajes
  2. Actividades
  3. Estados

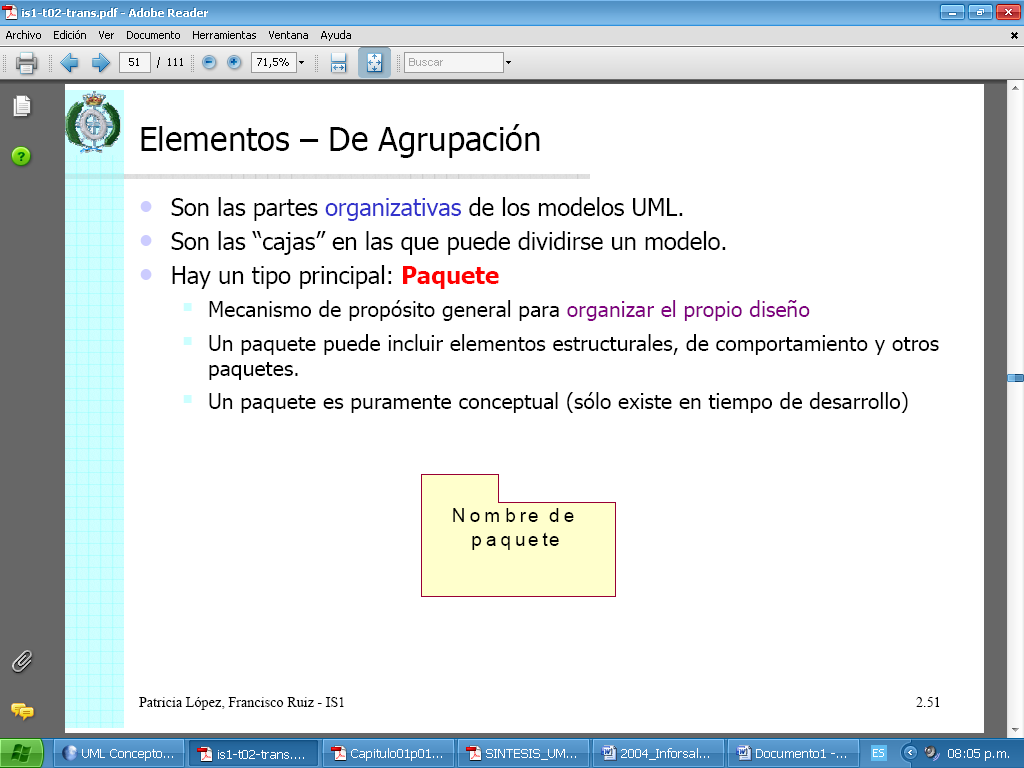
## **Elementos de Agrupación**

Los elementos de agrupación son las partes organizativas de los modelos UML. Estos son las cajas en las que se puede descomponer un modelo. Hay un tipo principal de elementos de agrupación.

* 1. Paquete

Un paquete es un mecanismo de propósito general para organizar el propio diseño. Un paquete puede incluir elementos estructurales, de comportamiento y otros paquetes.

Un paquete es puramente conceptual, sólo existe en la etapa de desarrollo. Gráficamente, un paquete se visualiza como una carpeta, normalmente incluye su nombre, y su contenido. Hay variaciones, tal como Frameworks, modelos y subsistemas (tipos de paquetes).



## **Elementos de Anotación**

A continuación se verán una serie de elementos notaciones pero sin entrar en detalle en la esencia de lo que es el concepto de cada uno de ellos.

También visualizaremos como representar gráficamente los conceptos principales de la orientación a objetos.

* 1. Nota.

Se puede colocar en los diagramas de activides para poder especificar mejor una actividad específica.

## **Elementos de Notación**

A continuación se verán una serie de elementos notaciones pero sin entrar en detalle en la esencia de lo que es el concepto de cada uno de ellos.

También visualizaremos como representar gráficamente los conceptos principales de la orientación a objetos. A continuación listaremos los siguientes elementos que serán tratados:

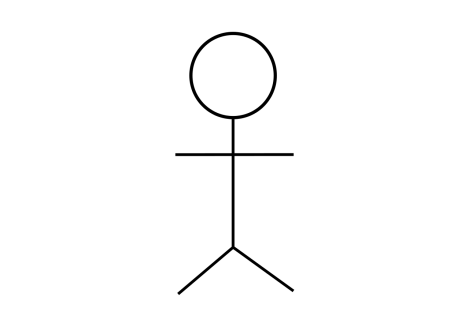
* 1. Nota.
  2. Dependencia.
  3. Clase.
  4. Objetos.
  5. Atributos.
  6. Métodos.
  7. Asociaciones.
  8. Dependencias.
  9. Responsabilidades.
  10. Navegabilidad.
  11. Composición.
  12. Relaciones.
  13. Herencia.
  14. Generalización.
  15. Actores.
  16. Relaciones.
  17. Caso de uso.
  18. Línea de vida.
  19. Activación.
  20. Mensaje.

## **Diagrama de caso de uso**

Es la representación gráfica de cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.  El caso de uso es la representación gráfica de cada proceso desde la perspectiva usuario, lo que se espera del sistema en cuanto a funcionalidad.

* 1. Actor

Representa un rol que es llevado a cabo por una persona, otro sistema, sensores, etc. El actor es representado por una figura en forma de persona.



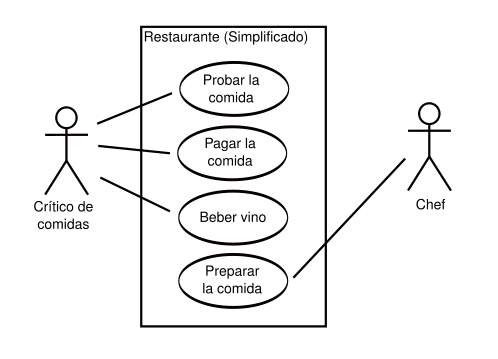
* 1. Relaciones

Este elemento representa la relación que existe entre un Caso-Uso y un Actor, dicho elemento es representado por una línea recta que se extiende de la figura del actor hacia el ovalo del caso-uso.

Entre dos casos de uso puede haber relaciones:

1. Extiende: Cuando un caso de uso especializa a otro extendiendo su funcionalidad.
2. Usa: Cuando un caso de uso utiliza a otro.

Se representa como una línea que une a los dos casos de uso relacionados, con una flecha en forma de triángulo y con una etiqueta “extiende” o “usa” según sea el tipo de relación.



* 1. Caso de Uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Se representa mediante un ovalo.

* 1. Límite de Sistema

Empleado para delimitar los límites del sistema.  Se representa por un rectángulo con color de fondo distintivo.

* 1. Escenario

Es el contexto del diagrama de caso de uso ya que nos permite mostrar un momento particular en la cual un conjunto de actores particular interactúan con el sistema.

* 1. Extends

Es otra forma de interacción, un caso de uso dado (la extensión) puede extender a otro. Esta relación indica que el comportamiento del caso de la extensión se utiliza en casos de uso, un caso de uso a otro caso siempre debe tener extensión o inclusión. El caso de uso extensión puede ser insertado en el caso de uso extendido bajo ciertas condiciones. La notación, es una flecha de punta abierta con línea discontinua, desde el caso de uso extensión al caso de uso extendido, con la etiqueta «extend».

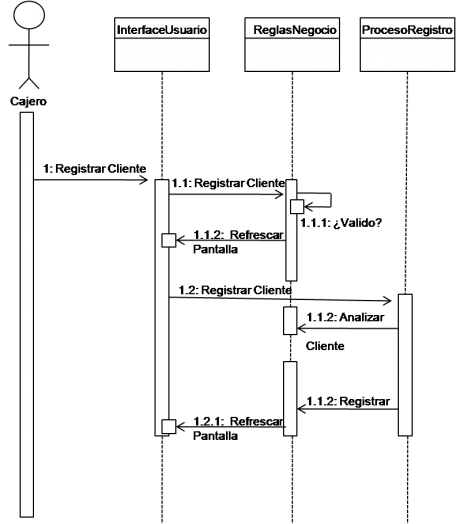
Esto puede ser útil para lidiar con casos especiales, o para acomodar nuevos requisitos durante el mantenimiento del sistema y su extensión. "La extensión, es el conjunto de objetos a los que se aplica un concepto. Los objetos de la extensión son los ejemplos o instancias de los conceptos." documentan el comportamiento de un sistema desde el punto de vista de un usuario en otras palabras será utilizado cuando un caso de uso sea similar a otro pero con ciertas variaciones, un ejemplo claro es que se necesite comprar azúcar y podemos seleccionar de entre azúcar rubia, blanca o su unidad de medida bolsa , kilo, etc.

* 1. Include

Es una forma de interacción o creación, un caso de uso dado puede "incluir" otro caso de uso. El primer caso de uso a menudo depende del resultado del caso de uso incluido. Esto es útil para extraer comportamientos verdaderamente comunes desde múltiples casos de uso a una descripción individual (si el actor realiza el caso de uso base tendrá que realizar también el caso de uso incluido), desde el caso de uso. El estándar de [Lenguaje de Modelado Unificado](https://es.wikipedia.org/wiki/UML) de [OMG](https://es.wikipedia.org/wiki/Object_Management_Group) define una notación gráfica para realizar diagramas de casos de uso, pero no el formato para describir [casos de uso](https://es.wikipedia.org/wiki/Casos_de_uso). Mucha gente sufre la equivocación pensando que un caso de uso es una notación gráfica (o es su descripción). Mientras la notación gráfica y las descripciones esto no sirve.

## **Diagrama de secuencia**

Muestra una interacción ordenada según la secuencia temporal de eventos.  El diagrama de secuencia muestra la interacción entre los objetos que tiene lugar a través del intercambio de mensajes, en particular, muestra los objetos participantes en la interacción y los mensajes que se intercambian ordenadas según la secuencia de tiempo.



* 1. Línea de vida de un objeto

Un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos. El rectángulo de encabezado contiene el nombre del objeto y el de su clase, en un formato nombreObjeto: nombreClase

* 1. Objeto activo

Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto.

* 1. Mensaje

El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta.

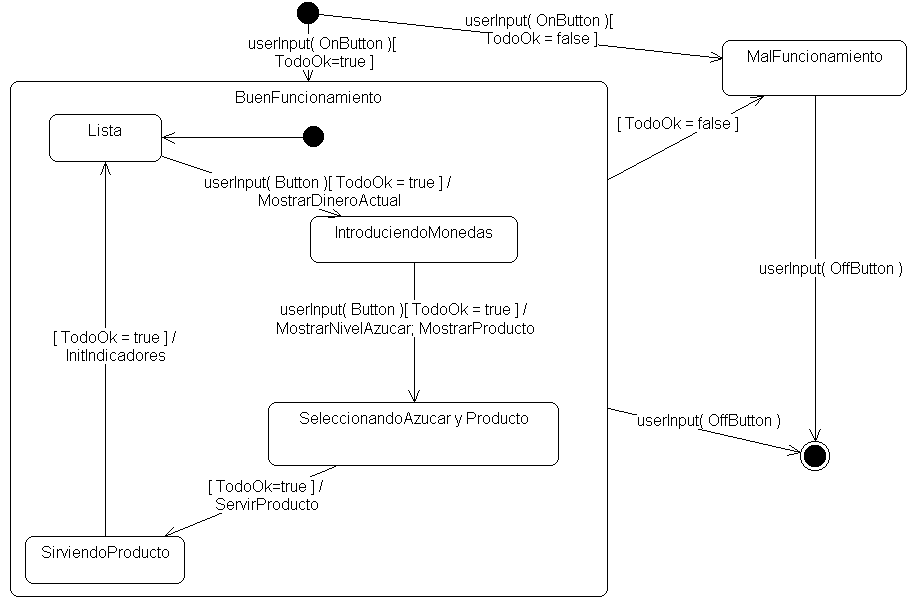
Los diagramas de clases y los objetos representan información estática, no obstante en un sistema funcional los objetos interactúan entre sí, y tales interacciones suceden con el tiempo, el diagrama de secuencias UML muestra la mecánica de interacción con base en tiempos.

* 1. Actor exterior

Entidad externa.

## **Diagrama de Estados**

Muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro. Un ejemplo en el caso de la cafetera son los estados posibles para la clase MaquinaCafe:



* 1. Estado

Identifica un periodo de tiempo del objeto (no instantáneo) en el cual el objeto esta esperando alguna operación, tiene cierto estado característico o puede recibir cierto tipo de estímulos.

* 1. Transición

Flujo de evento disparador, condición de guarda y acción.

* 1. Evento disparador

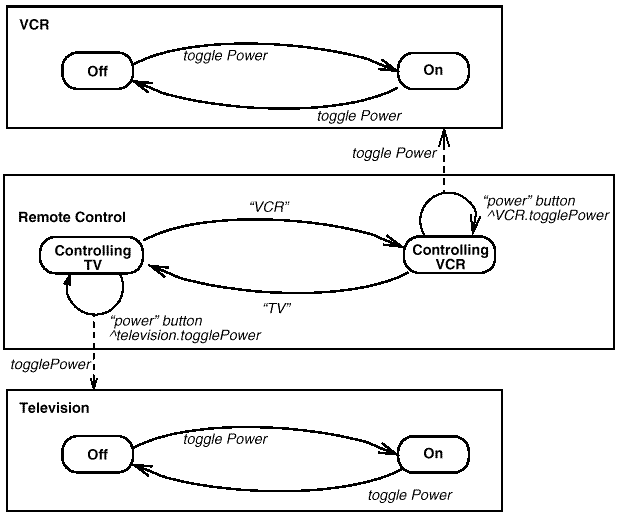
Por ejemplo recibir pedido, reparar copia.

* 1. Acción

Método que se ejecuta.

* 1. Condición de guarda

Condición de if.



## [**Diagrama de Actividades**](http://webdocs.cs.ualberta.ca/~pfiguero/soo/uml/actividades01.html)

Un diagrama de actividades es un caso especial de un diagrama de estados en el cual casi todos los estados son estados de acción (identifican que acción se ejecuta al estar en él) y casi todas las transiciones son enviadas al terminar la accion ejecutada en el estado anterior.

* 1. Actividad

Acción a realizarse.

* 1. División Sincronización

Actividades que deben ocurrir de manera sincronizada.

* 1. Transición de terminación

Conexión de una actividad a otra.

* 1. Unión Sincronización

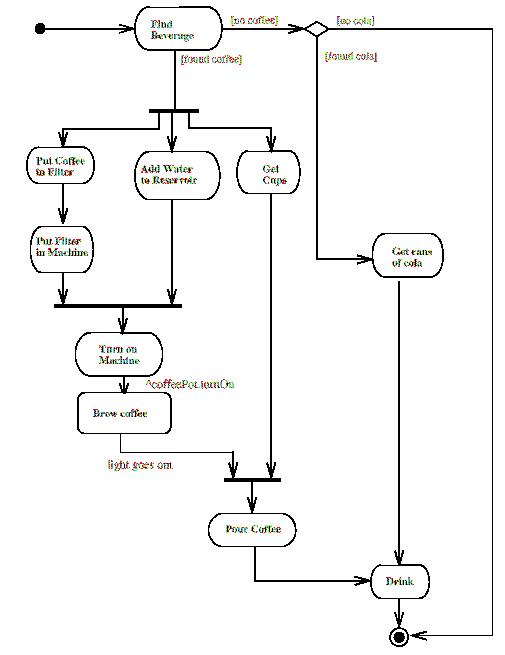
Actividades que después de ocurrir de forma sincronizada realizan una actividad en común.

* 1. Bifurcación o Reunificación

Condicionales que envían o reciben mensajes.

* 1. Calles y Flujo de objetos

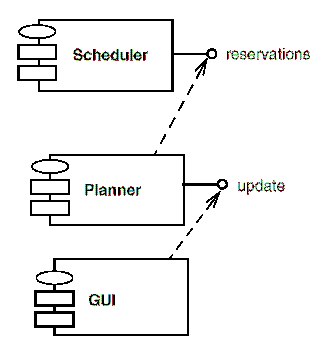
Se crean recuadros en los cuales se divide Cliente, Ventas, Almacén.



## **Diagrama de Componentes**

Es una unidad física de implementación con interfaces bien definidas pensada para ser utilizada como parte reemplazable de un sistema. Cada componente incorpora la implementación de ciertas clases del diseño del sistema. Cada componente incorpora la implementación de ciertas clases del diseño del sistema

Se representa como un grafo de componentes software unidos por medio de relaciones de dependencia (generalmente de compilación). Puede mostrar también contenencia de entre componentes software e interfaces soportadas. Un ejemplo es el siguiente:



En este caso tenemos tres componentes, GUI dependiendo de la interfaz update provista por Planner, Planner dependiendo de la interfaz reservations provista por Scheduler.

* 1. Interfaz

Muestra la organización y dependencia entre un conjunto de componentes.

* 1. Puerto

Especifica los servicios que uno de los componentes requiere.

* 1. Artefacto

En el se sitúan librerías, tablas, archivos, documentos, ejecutable.

* 1. Componentes

Es una parte física de un sistema (módulos, base de datos, programa ejecutable, etc).

* 1. Paquetes o subsistemas

Los componentes se pueden agrupar en paquetes.

* 1. Relaciones de dependencia, generalización, asociación y realización
  2. Nodo

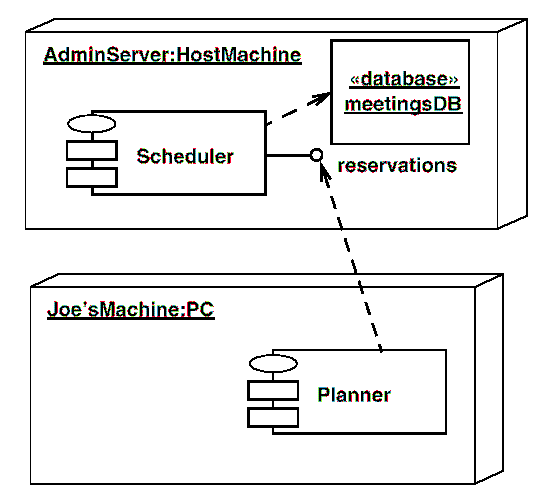
Representa un proceso o dispositivo en el cual se pueden desplegar los componentes.

## **Diagrama de Ejecución**

Un diagrama de ejecución muestra la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software, procesos y objetos que se ejecutan en ellos. Instancias de los componentes software representan manifestaciones en tiempo de ejecución del código. Componentes que solo sean utilizados en tiempo de compilación deben mostrarse en el diagrama de componentes.

Un diagrama de ejecución es un grafo de nodos conectados por asociaciones de comunicación. Un nodo puede contener instancias de componentes software, objetos, procesos (un caso particular de un objeto). Las instancias de componentes software pueden estar unidos por relaciones de dependencia, posiblemente a interfaces.

Un ejemplo de diagrama de ejecución es el siguiente:



En este caso se tienen dos nodos, AdminServer y Joe'sMachine. AdminServer contiene la instancia del componente Scheduler y un objeto activo (proceso) Denominado meetingsDB. EnJoe'sMachine se encuentra la instancia del componente software Planner, que depende de la interfaz reservations, definida por Scheduler.

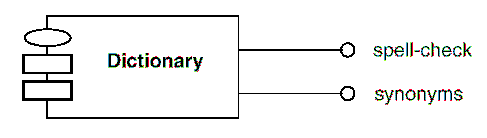
* 1. Nodos

Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Pueden representarse instancias o tipos de nodos. Se representa como un cubo 3D en los diagramas de implementación.

* 1. Componentes

Un componente representa una unidad de código (fuente, binario o ejecutable) que permite mostrar las dependencias en tiempo de compilación y ejecución. Las instancias de componentes de software muestran unidades de software en tiempo de ejecución y generalmente ayudan a identificar sus dependencias y su localización en nodos.  Pueden mostrar también que interfaces implementan y qué objetos contienen. Su representación es un rectángulo atravesado por una elipse y dos rectángulos más pequeños.

Un ejemplo de componente que implementa dos interfaces es



## **Diagrama de Clases**

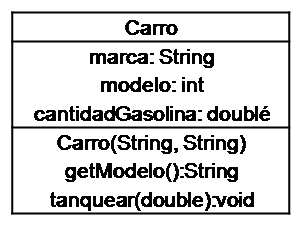
Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia y de uso. Muestra los bloques de construcción de cualquier sistema orientado a objetos. Los diagramas de Clase son más útiles para ilustrar relaciones entre clases e interfaces.

Un diagrama de clases está compuesto por los siguientes elementos:

* [Clase](http://users.dcc.uchile.cl/~psalinas/uml/modelo.html#clase): atributos, métodos y visibilidad.
* [Relaciones](http://users.dcc.uchile.cl/~psalinas/uml/modelo.html#relacion): Herencia, Composición, Agregación, Asociación y Uso.
  1. Clase

Una clase se representan como un rectángulo dividido en tres partes (tres compartimentos).

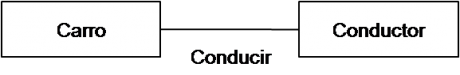
* En el compartimento superior, centrado, va el nombre de la clase.
* El segundo compartimento contiene los atributos, en la forma: visibilidad nombreAtributo: tipo
* El tercer compartimento contiene los métodos, en la forma: visibilidad nombreMétodo(parámetros): tipoRetorno.
* Los parámetros van separados por comas y sólo se especifica el tipo.
* Los constructores no tienen tipo de retorno.
* La visibilidad, de atributos y métodos, puede ser:  – privado, + público, # protegido, ~ de paquete



Ejemplo de clase

* 1. Asociaciones

Una asociación (relación entre dos clases) se representa como una línea continua entre dos Clases, y puede tener el nombre de la relación sobre esta línea. Ejemplo:

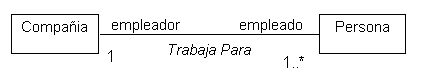
[](http://3.bp.blogspot.com/-fiUZI3Kazyk/TyNOLf0RuxI/AAAAAAAAAwg/nGFTGwyqqyU/s1600/ejemploasociacionconducir.png)

Ejemplo de asociaciones

En el ejemplo anterior cada Carro tiene una relación con Conductor, pero también cada Conductor tiene relación con Carro. Esta es una relación en ambos sentidos.

1. Asociación binaria

Se identifica como una línea sólida que une dos clases. Representa una relación de algún tipo entre las dos clases, no muy fuerte ( es decir, no se exige dependencia existencial ni encapsulamiento).



Un ejemplo es la relación entre una compañía y sus empleados

En este caso la relación recibe el nombre genérico Trabaja Para, la compañía tiene uno o más instancias de la clase Persona denominadas empleado y cada empleado conoce su empleador (en este caso único).

1. Sentido

Para mostrar que la relación sólo tiene un sentido se muestra con una flecha que indica el sentido de la relación. Ejemplo:

[http://3.bp.blogspot.com/-41X-hGO6xZo/TyNOLM3sc0I/AAAAAAAAAwU/Dc6Dta2lGKA/s1600/ejemploasociacion2.png](http://3.bp.blogspot.com/-41X-hGO6xZo/TyNOLM3sc0I/AAAAAAAAAwU/Dc6Dta2lGKA/s1600/ejemploasociacion2.png)

En este ejemplo un Pasajero conoce el Carro(o carros) con el cual viaja, pero el Carro no tiene ninguna relación con los Pasajeros.

1. Multiplicidad

Es una restricción que se pone a una asociación, que limita el número de instancias de una clase que pueden tener esa asociación con una instancia de la otra clase.

[http://3.bp.blogspot.com/-DrrJUVYz7BU/TyNOJozrTeI/AAAAAAAAAvg/On53-yx69WM/s1600/ejemplomultiplicidad.png](http://3.bp.blogspot.com/-DrrJUVYz7BU/TyNOJozrTeI/AAAAAAAAAvg/On53-yx69WM/s1600/ejemplomultiplicidad.png)

En este caso las relaciones son:

- Un chofer tiene relación con cero o más autobuses.

- Un autobús tiene relación con uno o dos choferes.

- Una terminal de pasajero tiene relación con cero o muchos autobuses.

- Un autobús tiene relación con un terminal de pasajero.

1. Roles

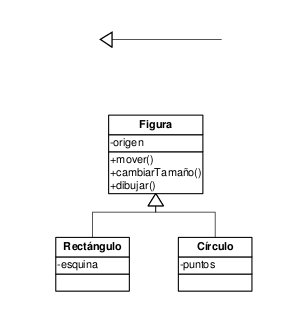
Para indicar el papel que juega una clase en una asociación se puede especificar un nombre de rol.  Se representa en el extremo de la asociación junto a la clase que desempeña dicho rol.

ejemploroles

Ejemplo de roles.

* 1. Generalización

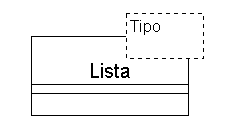
La relación de generalización denota una relación de herencia entre clases. Se representa dibujando un triángulo sin rellenar en el lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.



* 1. Clase paramétrica

Una clase paramétrica representa el concepto de clase genérica en los conceptos básicos OO o de témplate en C++. Se dibuja como una clase acompañada de un rectángulo en la esquina superior derecha, con los parámetros del caso.

Por ejemplo, la clase Lista que utiliza un parámetro formal Tipo se vería de la siguiente manera



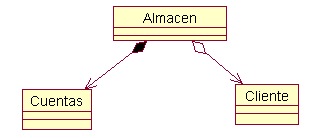
* 1. Agregación

http://users.dcc.uchile.cl/~psalinas/uml/img/modelo/agregacion1.jpg

Para modelar objetos complejos, n bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

* + - **Por Valor**: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
    - **Por Referencia**: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

* Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
* Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
* La composición (por Valor) se destaca por un rombo relleno.
* La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

* 1. Composición

Es una asociación fuerte, que implica tres cosas:

1. Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.
2. Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene
3. Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación.

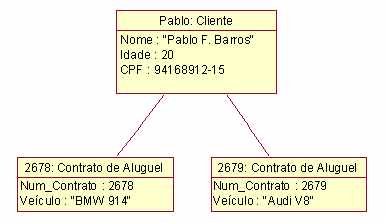
Existe también una relación de composición menos fuerte (no se exige dependencia existencial, por ejemplo) que es denotada por una un rombo sin rellenar en uno de los extremos.

## **Diagrama de Objetos**

Es una representación gráfica de la vista de diseño estática o de la vista de interacción estática de un sistema.

Los diagramas de objetos generalmente contienen:

* 1. Objetos
  2. Enlaces

****

* 1. Clases

## **Características de Diagrama de Objetos**

* 1. Los objetos se representan etiquetando el nombre de la instancia seguido de dos puntos (:) frente al nombre de la clase.
  2. Se centra en comunicar un aspecto de la vista de diseño estática o la vista de procesos estática de un sistema.
  3. Representa una escena de la historia representada por un diagrama de interacción.
  4. Contiene solo aquellos elementos esenciales para comprender ese aspecto.
  5. No es minimalista, no deja de informar al lector sobre la semántica importante.
  6. Los diagramas de objetos `pueden contener notas y restricciones.
  7. A veces se colocarán clases en los diagramas de objetos, cuando se quiera mostrar la clase que hay detrás de cada instancia.

## **Técnicas comunes de modelado de Objetos**

* 1. Modelado de estructuras de objetos

Si se congela un sistema en ejecución o uno se imagina un instante concreto en un sistema modelado, aparecerá un conjunto de objetos, cada uno de ellos en un estado específico, y con unas relaciones particulares con otros objetos. Los diagramas de objetos se pueden usar para visualizar, especificar, construir y documentar la estructura de esas instantáneas. Los diagramas de objetos son especialmente útiles para modelar estructuras complejas de datos.

Con los diagramas de objetos no se pueden especificar completamente la estructura de objetos del sistema. Puede existir una multitud de posibles instancias de una clase particular, y para un conjunto de clases con relaciones entre ellas, pueden existir muchas más configuraciones posibles de esos objetos. Por lo tanto, al utilizar diagramas de objetos solo se pueden mostrar significativamente conjuntos interesantes de objetos concretos o prototípicos. Esto es lo que significa modelar una estructura de objetos.

* 1. Ingeniería Inversa

Hacer ingeniería inversa (creación de un modelo a partir del código) con un diagrama de objetos puede ser muy útil. De hecho, mientras se está depurando el sistema, esto es algo que el programador o las herramientas están haciendo continuamente. Por ejemplo, si se persigue un enlace perdido, uno debe dibujar mental o literalmente un diagrama con los objetos afectados para saber dónde se invalida,

* 1. Modelado de Instancias discretas

Una clase estructurada puede utilizarse para modelar estructuras de datos en las que las partes tienen conexiones contextuales que solo se aplican dentro de ella. Los atributos o las asociaciones normales pueden definir partes compuestas de una clase, pero puede que las partes no estén relacionadas en un diagrama de clases plano. Una clase cuya estructura interna se muestra con partes y conectores evita este problema.

## **Clase Estructurada**

* 1. Modelado de una API

Los desarrolladores que ensamblan un sistema a partir de partes componentes a menudo desean ver las interfaces de programación de aplicaciones (API) que se utilizan para enlazar las partes. Las API representan las líneas de separación programáticas de un sistema y se pueden modelar mediante interfaces y componentes.

Una API es sencillamente una interfaz realizada por uno o más componentes. Los desarrolladores realmente solo se preocupan de la propia interfaz. El componente que realice las operaciones de interfaz no es relevante siempre y cuando haya algún componente que las realice. Sin embargo, desde la perspectiva de la gestión de la configuración de un sistema, estas realizaciones son importantes porque hay que asegurar que cuando se publica una API hay alguna realización disponible que cumple las obligaciones de la API. Afortunadamente, con UML podemos modelas ambas perspectivas.

* 1. Para modelar una API
* Hay que identificar las líneas de separación del sistema y modelar cada una como una interfaz, recogiendo los atributos y operaciones que forman su frontera.
* Hay que exponer solo aquellas propiedades de la interfaz que son importantes para comprender dentro del contexto dado; en otro caso, hay que esconder sus propiedades, manteniéndolas en la especificación de la interfaz por referencia, si son necesarias.
* Hay que modelar la realización de cada API solo en tanto que sea importante para mostrar la configuración de una implementación específica.

## **Diagrama de Comunicación**

Un diagrama de comunicación anteriormente conocidos como diagramas de colaboración, modela las interacciones entre objetos o partes en términos de mensajes en secuencia. Los diagramas de comunicación representan una combinación de información tomada desde el diagrama de clases, secuencia, y diagrama de casos de uso describiendo tanto la estructura estática como el comportamiento dinámico de un sistema.

Para mantener el orden de los mensajes en un diagrama de comunicación, los mensajes son etiquetados con un número cronológico, y colocados cerca del enlace por el cual se desplaza el mensaje. Leer un diagrama de comunicación conlleva comenzar en el mensaje 1.0, y seguir los mensajes desde un objeto hasta el siguiente, sucesivamente.

Cada diagrama UML tiene su propio propósito específico y un solo diagrama no es suficiente por sí solo. Los mensajes se representan como una flecha etiquetada que indica la dirección del mensaje, utilizando una notación similar a la utilizada en los diagramas de secuencia.

* 1. Objeto activo
  2. Enlace
  3. Mensaje
  4. Multiobjeto
  5. Objeto pasivo
  6. Enlace transitorio

Persiste durante la interacción después se desecha

## **Diagrama de Despliegue**

Los diagramas de despliegue son los complementos de los diagramas de componentes que, unidos, proveen la vista de implementación del sistema. Elementos:

* 1. Componentes

Representados como una caja rectangular con dos protuberancias del lado izquierdo) y asociaciones. Es una parte física de un sistema (módulos, base de datos, programa ejecutable, etc).

.

* 1. Interfaz

Muestra la organización y dependencia entre un conjunto de componentes.

* 1. Puerto

Especifica los servicios que uno de los componentes requiere.

* 1. Artefacto

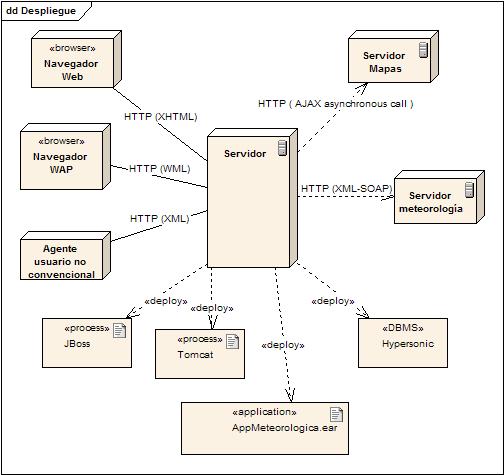
En el se sitúan librerías, tablas, archivos, documentos, ejecutable.

* 1. Paquetes o subsistemas

Los componentes se pueden agrupar en paquetes.

* 1. Relaciones de dependencia, generalización, asociación y realización
  2. Nodo

Representa un proceso o dispositivo en el cual se pueden desplegar los componentes.



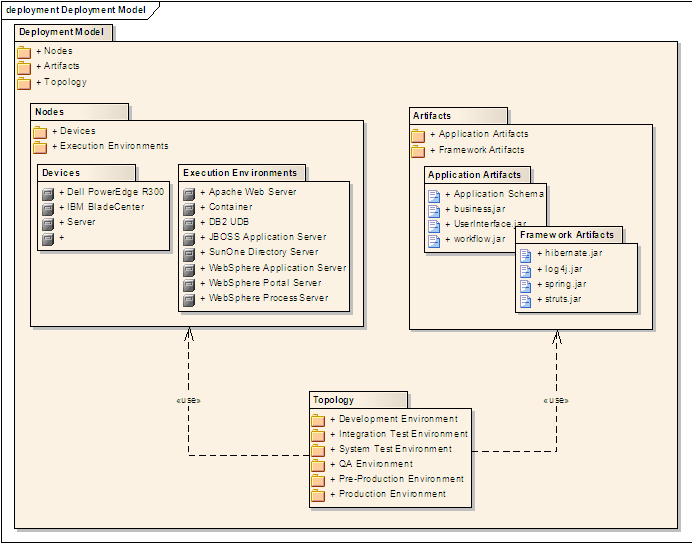
## **Diagrama de Paquetes**

En el Lenguaje Unificado de Modelado, un diagrama de paquetes muestra cómo un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones.

Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión. Cada paquete puede asignarse a un individuo o a un equipo, y las dependencias entre ellos pueden indicar el orden de desarrollo requerido. Elementos:

* 1. Paquetes
  2. Diagramas de UML internos

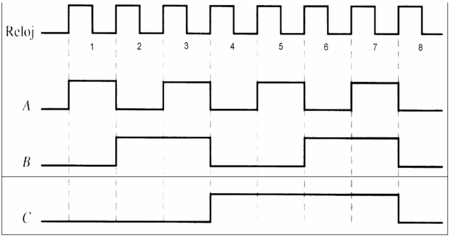


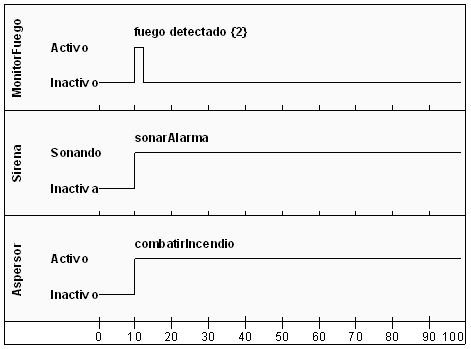
## **Diagrama de Tiempo**

Es un diagrama de interacción que muestra los tiempos reales entre diferentes objetos o roles, en oposición a la simple secuencia relativa de mensajes.

En el estándar de Lenguaje de Modelado Unificado de OMG los diagramas de tiempo son una representación especial de interacción que se enfoca en el tiempo de los mensajes enviados entre objetos.

* 1. Mensaje
  2. Objetos





## **Diagrama de Visión Global de Interacciones**

El diagrama global de las interacciones es un diagrama de comportamiento, más precisamente, uno de los cuatro diagramas de interacción. Muestra una cierta vista sobre los aspectos dinámicos de los sistemas modelados.

Es un hibrido entre un diagrama de actividades y un diagrama de secuencia

* 1. Actividad
  2. División Sincronización
  3. Transición de terminación
  4. Unión Sincronización
  5. Bifurcación o Reunificación
  6. Diagrama de Secuencia

